CMPSC-F265 Midterm Exam
Sample Practice

NAME:	Student ID:

Rules:

- 75 minutes, 100 possible points
- Closed book/notes
- No calculators, no electronic devices

Problem	Score
1	
2	
3	
4	
5	
6	
Total (100)	

Problem 1 Complexity Analysis and Big O Notation

1. Let *n* be the size of input, given the number of steps for different algorithms in the following table, write down the corresponding big O complexity. '^' indicates the exponential computation.

Number of steps	**Complexity order (big O)**
(n^2 + 1) / (n^2)	O(1)
(n+2)^3 + 2^n	$O(2^n)$
(log(4))^n + n*log(n)	$O(2^n)$

2 Determine the complexity order (big O) for each of the following codes

a)

```
for (int i=n/2; i< (n/2 + 2); i++) {
        //some statement
}
```

O(1)

b)

```
for (int i=1; i<n; i=i+2) {
        for (int j=n; j>1; j=j/2) {
                //some statement
        }
}
```

n/2*logn and therefore O(nlogn)

c)

```
for (int i=1; i<n; i++) {
        for (int j=1; j<i; j++) {
                //some statement
        }
        for (int k=1; k<n; k++) {
                //some statement
        }
}
```

$O(n^2)$

Problem 2 Stack, Queue, and Priority Queue

1. What is the output of the following code?

```
public static void main(String[] args)
        {
                Stack myStack new Stack(5);
                Queue myQueue = new Queue(5);

                for(int i=1; i<=5; i++)
                        myStack.push(i);

                myStack.display();//prints stack in a column from top downwards

                while(!myStack.isEmpty())
                        myQueue.enqueue(myStack.pop());
                while(!myQueue.isEmpty())
                        myStack.push(myQueue.dequeue());


                myStack.display();//prints stack in a column from top downwards
        }
```

Ans: 1 2 3 4 5

2. Suppose that a minus sign in the input indicates *dequeue* the priority queue, and other string indicates *enqueue* the string into a **priority queue**. What will be the content (from front to rear) of the priority queue with the following inputs. Suppose the priority of the strings is determined by their natural order (i.e. 'A'<"B' and 'B' < 'C')

B D - A E - - F C - - M P - O G Z K - -

For this PriorityQueue, "A" will be dequeued out before "B" determined by their "priority". And therefore, the result would be:

The letters remained in the PriorityQueue (from the front to the rear) would be:

P O Z

3. Consider an **array-based** implementation of Queue. Implement the **enqueue()** method.

class Queue

```
{
        private int maxSize;
        private int[] queArray;
        private int front;
        private int rear;
        private int nItems;
//--------------------------------------------------------------
        public Queue(int s) // constructor
        {
                maxSize = s;
                queArray = new int[maxSize];
                front = 0;
                rear = -1;
                nItems = 0;
        }
//--------------------------------------------------------------
    public void enqueue(int j) {
                //YOUR CODES
                // the front variable indicating the index of the next element that will be
dequeued out.
                // the rear variable indicating the index of the next new element when
enqueued will be placed at.

               If (rear ==  maxSize-1) rear= -1;   // this is to deal with the situation when
variable has reached the last index of the array.  We need to wraparound it to -1.

               queArray[++rear] = j;
               nItems++;

        }
```

4. Suppose you were asked to write a method that will take two sorted stacks A and B (min on top) and create one stack that is sorted (min on top). You are allowed to use only the stack operations such as pop, push, size and top. No other data structure such as arrays are not allowed. You are allowed to use stacks. Note that elements on the stack are comparable.

```
public Stack mergedSortedStacks (Stack A, Stack B){
        // YOUR CODES
        // top() here is just the method as peek()
        // suppose it is a Stack of Integers.

        Stack<Integer> resultStack = new Stack<Integer>();
        // when both stacks are not empty, push the one with smaller values into
resultStack
```

```
        while (!A.isEmpty() && !B.isEmpty()){
                If (A.peek()<B.peek()) resultStack.push(A.pop());
              else  resultStack.push(B.pop());
        }

        // add remaining elements from stack A to the resulting stack.
        while (!A.isEmpty()) {
                resultStack.push(A.pop());
        }

        // add remaining elements from stack B to the resulting stack.
        while (!B.isEmpty()) {
                resultStack.push(B.pop());
        }

}
```

Problem 3. Sorting Algorithms

1. Given the following array:

   7 0 5 4 3 6 6 9

   a) Show the content of the array for the first two iterations using Seletion sort to sort it.

      0 7 5 4 3 6 6 9
      0 3 5 4 7 6 6 9

   b) Show the content of the array for the first two iterations using Insertion sort to sort it.

      0 7 5 4 3 6 6 9
      0 5 7 4 3 6 6 9

   c) Show the content of the array for the first two iterations using Bubble sort to sort it.

      0 5 4 3 6 6 7 9
      0 4 3 5 6 6 7 9

   d) Show the content of the array after the first partition using Quick sort (suppose

you choose the first element as Pivot).

6 0 5 4 3 6 7 9

2. Choose the best sorting algorithm in the following scenarios AND write it's time complexity in that scenario. Choose just among: insertion sort, merge sort, quick sort(pivot being the last item).

a) When the input is sorted.
The best case for insert: O(N), merge sort O(nlogn) and quick sort O(nlogn)
And therefore we can choose insertion sort

b) When the available memory is limited.

Quick sort, since it has the best average efficiency as compared to insertion sort and merge sort, and that it is an in-place algorithm.

3. Suppose the running time of applying Selection Sort on a given array is 1024ms. How much time do we need to spend using Merge sort to sort on the same array?

For Selection sort, the sorting efficiency is O(n^2)
So we know that N^2 = 2^10 = 1024, and so n = 2^5 = 32
For merge sort, it is O(nlogn), plug-in n=32, we have 32log32 = 32 * 5 = 160

4. Assume that we call partition on an array of size 10, and the result after partitioning is:

5	3	2	11	15	21	12	17	77	66

What was the pivot? 11 would be the pivot, as it is the only number that satisfies the requirement that its left subarray only contains numbers equal or smaller than the pivot, and its right subarray only contains numbers equal to or greater than the pivot.

5. Assume I have an efficient method to compute median of an array in O(n). In this case, if I choose the median as the pivot, what would be the time complexity of quick sort?

This would be the best case for quick sort, in which we can always choose the middle element as the pivot. It would still be O(nlogn) (It is the average case efficiency for Quick sort).

You can also achieve this by analyzing the recurrence relation:
T(n) = 2T(n/2)+ O(n)+ O(n)

The first O(n) is to do partition, and the second O(n) is to find the median of the entire array. To solve this recurrence relation, T(n) would still be O(nlogn).

6. Consider the partition method in Quick sort (the pivot is the last item), complete the missing parts of the partition method (underlined in the following code).

```
public int partition(int arr[], int low, int high)
   {
      int pivot = arr[high];
      int i  = (low-1);
      for (int j=low; _____j<high_____ ; j++)
      {
         if (_____a[j]<=pivot_____)
         {
            i++;
            // swap arr[i] and arr[j]
            int temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
         }
      }

      int temp = arr[i+1];
      arr[i+1] = arr[high];
      arr[high] = temp;

      return ____i+1_____;
   }
```

Problem 4. Linked List

1. Fill the following table with time complexities of different operations in the specified data structures.

	insertFirst()	**insertLast()**	**deleteLast()**
Singly linked list	O(1)	O(n)	O(n)
Double-ended singly linked list	O(1)	O(1)	O(n)
Double-ended doubly linked list	O(1)	O(1)	O(1)

2. Assume that LL is a DOUBLY linked list with the head node and at least one other internal node M that is not the last node. Write few lines of code to accomplish the following. You may assume that each node has a *next* pointer and *prev* pointer. There is no other existing method for you to call. Note that for each operation, you need to manipulate at least two pointers, next and prev.

a). Return the one-to-the-last link in a double-ended doubly linked list. Write the return statement.

return last.prev;

b) Delete the head node
head.next.prev = null;
head = head.next;

c). Swap head and the M node (you are not allowed to simply swap the data)

//first introduce three temporary nodes references:
tempNode1 = m.prev;
tempNode2 = m.next;
tempNode3 = head.next;

// and then change the references;
tempNode1.next = head;

```
head.prev = tempNode1;
tempNode2.prev = head;
head.next = tempNode2;
m.prev = null;
m.next = tempNode3;
tempNode3.prev = m;
head=m;
```

3. Complete the following method in the single-ended singly LinkedList class. The method getCount is supposed to return the total number of nodes whose data is equal to the given value c. You can assume the Node class has the public fields, data (an integer) and next (a pointer to another Node). First is the pointer to the first node in the list. You have no other existing methods to call.

```
public int getCount(Node first, int c){

// YOUR CODES

int count = 0;
Node current  = first;
while (current!=null){
   If  (current.data==value) count++;
   current = current.next;
}
return count;
}
```

4. Complete the following method in the single-ended singly LinkedList class. The method insertAt returns nothing, but it will insert a new node with data *val* at position *pos* in the linked list. It is assumed the first node is at position 1. You have no other methods to call. Show the time complexity of this method.

```
public void insertAt(Node first, int val, int pos){
  Node newNode = new Node(val);

  if (pos==1) {
    // the newNode will be the first node.
    if (first==null) first = newNode;
    else {
      newNode.next = first.next;
      first = newNode;
    }
  } else {
    Node current = first;
    Node previous = first;
    int count = 1;
    // Find the proper place to insert the newNode
    // if pos is larger than the linked list size, insert the new node at the end.
    while (current!=null && count<pos){
      previous = current;
      current = current.next;
      count++;
    }
    previous.next = newNode;
    newNode.next = current;
  }
}
```

5. Complete the following method in the single-ended singly LinkedList class. The method deleteAt returns nothing, but it will delete the node at position pos from the linked list. It is assumed the first node is at position 1. You have no other methods to call. Show the time complexity of this method.

```
public void deleteAt(Node first, int pos){
  int size=0;
  Node current = first;
  while (current!=null){
    current = current.next;
    size++;
  }

  if (first==null || pos>size) return;
  if (pos==1) {
    first =  first.next;
  } else{
    Node previous = first;
    Node current = first;
    int count = 1;
    while (current.next!=null && count<pos){
      previous = current;
      current = current.next;
      count++;
    }
    previous.next = current.next;
  }
}
```

Problem 5 Recursion

1) Write the recurrence relation(T(n)) for the foo() method below.

```
public static void foo(int low, int high) {
   if (low <= high) {   // It should be <= here not !=.  Sorry about this.
       int mid = (low + high) / 2;
       foo(low, mid – 1);
       System.out.println(mid);
       foo(mid + 1, high);
   }
}
```

$T(n) = 2 + 2T(n/2)$

2) What is the result for foo(0,4)?

0
1
2
3
4

2. What will be the value returned when calling the following recursive method by mstery(14, 10)

```
public int mystery(int x, int y){
     if (x<y) {
        return x;
     } else {
        return mystery(x-y, y);
     }
}
```

```
mstery(14, 10)
  mstery(4, 10)   // this will return 4 and then finally return 4.
```

3. Tribonacci numbers are similar to Fibonacci numbers, but instead of adding the two previous numbers to get the next number, we need to add the three previous numbers. The sequence is like: 0, 0, 1, 1, 2, 4, 7, 13, 24, … That means Trib(0)=0, Trib(1)=0, Trib(2)=1.

Write a recursive method to find the nth Tribonacci number:

```
public int trib(int n)  {
// YOUR CODES
if (n==0) return 0;
if (n==1) return 0;
if (n==2) return 1;
return trib(n-3)+trib(n-2)+trib(n-1);
```

```
}
```

Problem 6: Others

1. What is the best data structure to solve the following problem? A list needs to be built dynamically. Data must be easy to find, preferably in O(1). The user does not care about any order statistics such as finding max or min or median.

i. Use an Array

ii. Use a Singly LL

iii. Use a Stack

iv. Use a Queue

v. None of the above

The answer would be v: None of the above.

2. The Comparable and Comparator Interface

Implement a comparable data type **Employee** that represents an employee, each characterized by a *firstname* (String), *lastname* (String), *employeeID* (String), *department* (String), *age* (int) and salary (double). Your data type must support the following API:

Method/class	**Description**
Employee(String id, String firstname, String lastname, String dept, int age, double salary)	Construct an employee given his/her id, firstname, lastname, department, age and salary.
boolean equals(Employee that)	Is this employee's id is the same as that employee's id?
String toString()	A string representation in the format of "id, firstname, lastname"
int compareTo(Employee that)	Compare this employee with that employee by their lastname
static class AgeOrder	A comparator comparing two employees in their ages (older employee comes first)
static class SalaryOrder	A comparator comparing to employees their salaries (higher salary comes first)

```
import jave.util.Comparator;

public class Employee implements Comparable<Employee>{
  private String id;
  private String firstname;
  private String lastname;
  private String dept;
  private int age;
  private double salary;

……

  public boolean equals(Employee that){
    //YOUR CODES
return this.id.equals(that.id);

  }

  public String toString(){
    //YOUR CODES

  return id + " " + firstname + " " + lastname;

  }

  public int compareTo(Employee that){
    // YOUR CODES
    return this.lastname.compareTo(that.lastname);
  }

  public static class AgeOrder implements Comparator<Employee>{
      // YOUR CODES

          public int compare(Employee e1, Employee e2){
              if (e1.age>e2.age) return -1;
              else if (e1.age<e2.age) return 1;
              else return 0;
          }
  }
```

```
public static class  SalaryOrder implements Comparator<Employee>{
  // YOUR CODES

      public int compare(Employee e1, Employee e2){
            if (e1.salary>e2.salary) return -1;
            else if (e1.salary<e2.salary) return 1;
            else return 0;

      }

}
```

Suppose employees is an array of Employee objects. Write a statement that uses Arrays.sort() to sort employees by their salary.

Your answer: Arrays.sort(employees, new SalaryOrder ());